Lab-04

1 - Basic Queue Implementation

```
import json
import pickle
import requests
from datetime import datetime, timedelta
from typing import Optional, Any, Dict, List

# Part 1: Basic Queue Implementation
class Queue:

    def __init__(self):
        self._items = []

    def insert(self, value: Any) -> None:
        self._items.append(value)

    def pop(self) -> Optional[Any]:
        if self.is_empty():
            print("Warning: Attempted to pop from an empty queue")
            return None
        return self._items.pop(0)

    def is_empty(self) -> bool:
        return len(self._items) == 0

    def size(self) -> int:
        return len(self._items)

    def __str__(self) -> str:
        return f"Queue({self._items})"
```

```
if __name__ == "__main__":
    print("=== Testing Basic Queue ===")
    q1 = Queue()
    print(f"Is empty: {q1.is_empty()}")

    q1.insert("First")
    q1.insert("Second")
    q1.insert("Third")
    print(f"Queue: {q1}")

    print(f"Popped: {q1.pop()}")
    print(f"Popped: {q1.pop()}")
    print(f"Queue after pops: {q1}")

    print(f"Popped: {q1.pop()}")
    print(f"Trying to pop from empty queue: {q1.pop()}")
```

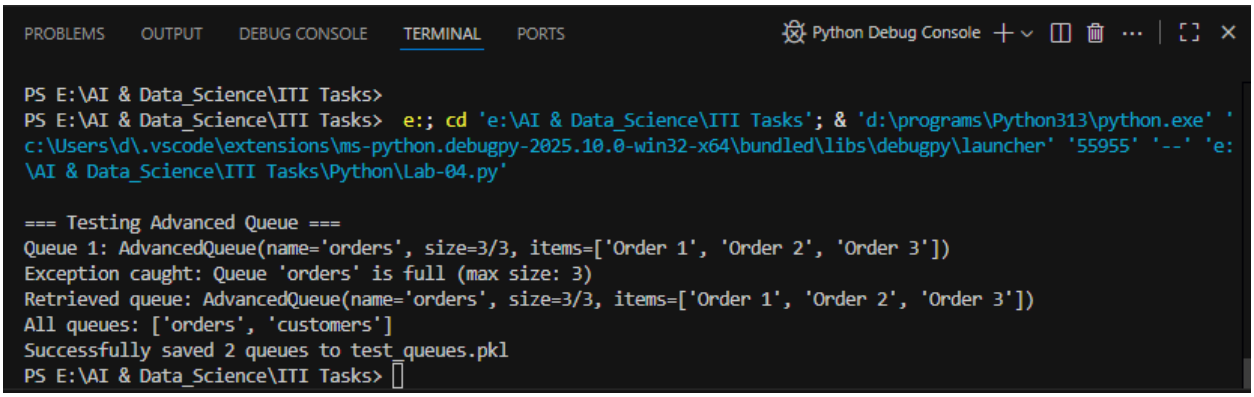
2 - Advanced Queue Implementation

```
class QueueOutOfRangeException(Exception):
    pass

class AdvancedQueue:
```

```
_instances: Dict[str, 'AdvancedQueue'] = {}

def __init__(self, name: str, max_size: int):

    self.name = name
    self.max_size = max_size
    self._items = []

    AdvancedQueue._instances[name] = self

def insert(self, value: Any) -> None:
    if len(self._items) >= self.max_size:
        raise QueueOutOfRangeException(
            f"Queue '{self.name}' is full (max size: {self.max_size})"
        )
    self._items.append(value)

def pop(self) -> Optional[Any]:
    if self.is_empty():
        print(f"Warning: Attempted to pop from empty queue '{self.name}'")
        return None
    return self._items.pop(0)

def is_empty(self) -> bool:
    return len(self._items) == 0

def size(self) -> int:
    return len(self._items)

@classmethod
def get_queue(cls, name: str) -> Optional['AdvancedQueue']:
    return cls._instances.get(name)

@classmethod
def list_all_queues(cls) -> List[str]:
```

```
        return list(cls._instances.keys())

    @classmethod
    def save(cls, filename: str = "queues.pkl") -> None:
        try:
            queue_data = {}
            for name, queue in cls._instances.items():
                queue_data[name] = {
                    'name': queue.name,
                    'max_size': queue.max_size,
                    'items': queue._items.copy()
                }

            with open(filename, 'wb') as f:
                pickle.dump(queue_data, f)
            print(f"Successfully saved {len(queue_data)} queues to {filename}")
        except Exception as e:
            print(f"Error saving queues: {e}")

    @classmethod
    def load(cls, filename: str = "queues.pkl") -> None:
        try:
            with open(filename, 'rb') as f:
                queue_data = pickle.load(f)

            cls._instances.clear()

            for name, data in queue_data.items():
                queue = cls(data['name'], data['max_size'])
                queue._items = data['items']

            print(f"Successfully loaded {len(queue_data)} queues from {filenam
e}")
        except FileNotFoundError:
            print(f"File {filename} not found")
        except Exception as e:
```

```
            print(f"Error loading queues: {e}")

    def __str__(self) -> str:
        return f"AdvancedQueue(name='{self.name}', size={len(self._items)}/{sel
f.max_size}, items={self._items})"


if __name__ == "__main__":
    print("\n=== Testing Advanced Queue ===")
    try:
        aq1 = AdvancedQueue("orders", 3)
        aq2 = AdvancedQueue("customers", 5)

        aq1.insert("Order 1")
        aq1.insert("Order 2")
        aq1.insert("Order 3")

        print(f"Queue 1: {aq1}")

        try:
            aq1.insert("Order 4")
        except QueueOutOfRangeException as e:
            print(f"Exception caught: {e}")

        retrieved_queue = AdvancedQueue.get_queue("orders")
        print(f"Retrieved queue: {retrieved_queue}")

        print(f"All queues: {AdvancedQueue.list_all_queues()}")



        AdvancedQueue.save("test_queues.pkl")

    except Exception as e:
        print(f"Error in advanced queue testing: {e}")
```



3 - Weather API Client

```
class WeatherAPIClient:
    """Client for Open-Meteo free weather API services."""

    def __init__(self):
        self.base_url = "https://api.open-meteo.com/v1"
        self.geocoding_url = "https://geocoding-api.open-meteo.com/v1"

    def _get_coordinates(self, city: str) -> Optional[tuple]:
        try:
            response = requests.get(f"{self.geocoding_url}/search",
                                    params={"name": city, "count": 1, "language": "en", "form
at": "json"})
            response.raise_for_status()
            data = response.json()

            if data.get("results") and len(data["results"]) > 0:
                result = data["results"][0]
                return (result["latitude"], result["longitude"])
            return None
        except requests.exceptions.RequestException as e:
            print(f"Geocoding request failed: {e}")
            return None
```

```
    def _make_weather_request(self, params: Dict[str, Any]) -> Optional[Dict]:
        try:
            response = requests.get(f"{self.base_url}/forecast", params=params)
            response.raise_for_status()
            return response.json()
        except requests.exceptions.RequestException as e:
            print(f"Weather API request failed: {e}")
            return None

    def get_current_temperature(self, city: str) -> Optional[float]:
        coordinates = self._get_coordinates(city)
        if not coordinates:
            print(f"Could not find coordinates for city: {city}")
            return None

        lat, lon = coordinates
        params = {
            "latitude": lat,
            "longitude": lon,
            "current_weather": "true",
            "timezone": "auto"
        }

        data = self._make_weather_request(params)
        if data and 'current_weather' in data:
            return data['current_weather']['temperature']
        return None

    def get_temperature_after(self, city: str, days: int, hour: Optional[int] = Non
e) -> Optional[float]:
        if days < 1 or days > 7:
            print("Days must be between 1 and 7 for the free API")
            return None

        coordinates = self._get_coordinates(city)
```

```
        if not coordinates:
            print(f"Could not find coordinates for city: {city}")
            return None

        lat, lon = coordinates

        target_date = datetime.now().date() + timedelta(days=days)

        if hour is not None:
            if not (0 <= hour <= 23):
                print("Hour must be between 0 and 23")
                return None

            params = {
                "latitude": lat,
                "longitude": lon,
                "hourly": "temperature_2m",
                "start_date": target_date.strftime("%Y-%m-%d"),
                "end_date": target_date.strftime("%Y-%m-%d"),
                "timezone": "auto"
            }

            data = self._make_weather_request(params)
            if data and 'hourly' in data:
                try:
                    hourly_temps = data['hourly']['temperature_2m']
                    return hourly_temps[hour] if hour < len(hourly_temps) else None
                except (KeyError, IndexError) as e:
                    print(f"Error parsing hourly data: {e}")
                    return None
        else:
            params = {
                "latitude": lat,
                "longitude": lon,
                "daily": "temperature_2m_max,temperature_2m_min",
                "start_date": target_date.strftime("%Y-%m-%d"),
```

```
            "end_date": target_date.strftime("%Y-%m-%d"),
            "timezone": "auto"
        }

        data = self._make_weather_request(params)
        if data and 'daily' in data:
            try:
                # Return average of max and min temperature
                max_temp = data['daily']['temperature_2m_max'][0]
                min_temp = data['daily']['temperature_2m_min'][0]
                return (max_temp + min_temp) / 2
            except (KeyError, IndexError) as e:
                print(f"Error parsing daily data: {e}")
                return None

    return None

  def get_lat_and_long(self, city: str) -> Optional[tuple]:
    return self._get_coordinates(city)

if __name__ == "__main__":
    print("\n=== Weather API Client Example (Open-Meteo Free API) ===")

  weather_client = WeatherAPIClient()

  try:
    print("\n1. Testing current temperature...")
    temp = weather_client.get_current_temperature("Cairo")
    if temp is not None:
        print(f"Current temperature in Cairo: {temp}°C")
    else:
        print("Failed to get current temperature")

    print("\n2. Testing coordinates...")
    coords = weather_client.get_lat_and_long("Cairo")
    if coords:
```

```
            print(f"Cairo coordinates: {coords}")
        else:
            print("Failed to get coordinates")

        print("\n3. Testing forecast...")
        forecast = weather_client.get_temperature_after("Cairo", 3)
        if forecast is not None:
            print(f"Temperature in Cairo in 3 days (daily average): {forecast:.1f}°
C")
        else:
            print("Failed to get forecast")

        print("\n4. Testing hourly forecast...")
        hourly_forecast = weather_client.get_temperature_after("Cairo", 2, 14)
        if hourly_forecast is not None:
            print(f"Temperature in Cairo in 2 days at 2 PM: {hourly_forecast:.1f}°
C")
        else:
            print("Failed to get hourly forecast")

    except Exception as e:
        print(f"Error testing weather API: {e}")
```


PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
Python Debug Console
PS E:\AI & Data_Science\ITI Tasks> e:; cd 'e:\AI & Data_Science\ITI Tasks'; & 'd:\programs\Python313\python.exe' '
c:\Users\d\.vscode\extensions\ms-python.debugpy-2025.10.0-win32-x64\bundled\libs\debugpy\launcher' '56096' '--' 'e:
\AI & Data_Science\ITI Tasks\Python\Lab-04.py'
=== Weather API Client Example (Open-Meteo Free API) ===
1. Testing current temperature...
Current temperature in Cairo: 27.8°C
2. Testing coordinates...
Cairo coordinates: (30.06263, 31.24967)
3. Testing forecast...
Temperature in Cairo in 3 days (daily average): 30.0°C
4. Testing hourly forecast...
Temperature in Cairo in 2 days at 2 PM: 33.7°C
PS E:\AI & Data_Science\ITI Tasks>